REMARKS

In response to the *Final Office Action* mailed May 5, 2006[1], Applicants respectfully request reconsideration. Claims 1-3, 5-12 and 14-21 remain pending in this application of which claims 1, 10, 19, and 20 are independent. With this Amendment, Applicants amend the specification to clarify statements of fact regarding the state of the prior art embodied by the [incr Tk] language, voluntarily amend independent claims 1, 10, 19, and 20, and add claims 23-26 to clarify elements of the Applicants' invention recited in the claims. In view of the foregoing amendments and the following remarks, Applicants respectfully traverse the Examiner's rejections of the claims under 35 U.S.C. § 102 and 35 U.S.C. § 103.

Double Patenting Rejection

In the *Final Office Action* mailed May 5, 2006, claims 1-3, 5-12, and 14-21 are rejected by the Examiner under the judicially created doctrine of obviousness-type double patenting as being respectively unpatentable over claims 1-12 of U.S. Patent Application Publication No. US 2002/0100033 A1 (US Application No. 09/759,697). Applicants request that the Examiner hold the double patenting rejection in abeyance until claims 1-3, 5-12, and 14-21 are otherwise allowable.

Rejections Under 35 U.S.C. § 102

In the *Final Office Action* mailed May 5, 2006, the Examiner rejected claims 1, 3, 5-6, 9-10, 12, 14-15, and 18-21 as being anticipated by McLennan, Michael J., "Object-oriented Programming with [incr Tcl] Building Mega-Widgets with [incr Tk]" hereinafter

---

[1] The Office Action may contain statements characterizing the related art, case law, and claims. Regardless of whether any such statements are specifically identified herein, Applicants decline to automatically subscribe to any statements in the Office Action.

McLennan. The Examiner objected to claim 1, incorporating the objection of claim 1 to independent claims 10, 19, and 20, and depending claims therefrom.

On pages 2-3, the Examiner objected to claim 1 contending that *McLennan* discloses notifying objects of a change in an option value through a change handler identified by an option binding, the option binding being located by first searching a mapping data structure for a previously computed mapping to the option binding and, if no mapping was previously computed, by then computing the mapping to the option binding and storing the mapping in the mapping data structure where the code for the change handler for the option is defined in different classes within a class inheritance hierarchy.

Applicants respectfully disagree. It is Applicants' position that *McLennan* does not disclose at least three elements of the invention that are recited in the claims:

- notifying the object, instantiated from a class *within a class inheritance hierarchy*, of a change in an option value by searching a mapping data structure;
- notifying the object through change handlers identified by an option binding located *by first searching a mapping data structure and, if no mapping was previously computed, by then computing and storing the mapping*; and
- *supporting references to option values without preallocation of memory space*.

Notifying the Object, Instantiated from a Class Within a Class Inheritance Hierarchy, of a Change in an Option Value by Searching a Mapping Data Structure

It is Applicants' position that *McLennan* does not disclose the notification of an object instantiated from a class within a class inheritance hierarchy by searching a mapping data structure. *McLennan*'s disclosed data structure (exemplified in Figure 2-8) does not facilitate executing change handlers for an instance object specifically created under a class inheritance hierarchy. The *McLennan* data structure associates multiple objects, linking objects together despite the objects being instances of separate classes unrelated by a class inheritance hierarchy. Therefore, the change handler in *McLennan* is a mechanism by which a change to a configuration option of a master "mega-widget" object can be propagated to component widgets that are linked, *but not necessarily related by a class inheritance hierarchy*.

In support of Applicants' contention, Applicants refer the Examiner to Example 2-15 (*McLennan*, p. 107) in which *McLennan* discloses a `Fileconfirm` class. The `Fileconfirm` class adds a `Fileviewer` component in its constructor. Adding a component to a class is not an example of inheritance. In Example 2-15, *McLennan* describes his option change handler as follows:

> When the user selects a file, the `Fileviewer` executes its -`selectcommand` code, which calls the `Fileconfirm::select` method with the selected file name substituted in place of the `%n`.

In the above example, it is evident that *McLennan* is using a data structure that associates objects, *unrelated by a class inheritance hierarchy*, in order to propagate option changes to component, as opposed to superclass, widgets.

In contrast, Applicants' claimed mapping data structure is a mechanism for efficiently identifying and notifying all the *right change handlers to execute for a single*

object, whose class type may have multiple superclasses defined by a class hierarchy, when an option value changes on that object.

Notifying the Object by First Searching a Mapping Data Structure and, if No Mapping was Previously Computed, by then Computing and Storing the Mapping

In further support of Applicants' position, *McLennan's* notification mechanism also does not enable notifying an object by first searching a mapping data structure for a previously computed mapping and, if no mapping was previously computed, by then computing the mapping to the option binding and storing the mapping in the mapping data structure.

In support of Applicants' contention, Applicants refer the examiner to the example on page 75, wherein *McLennan* discloses a method called `itk_initialize` that *must be called to make sure that all mega-widget options are properly initialized*. McLennan cautions that "*if you forget to call it for a particular class, some of the configuration options may be missing whenever you create a mega-widget of that class.*" (emphasis added). It logically follows that *McLennan* does not teach the computing of the mapping to an uninitialized option binding.

In contrast, Applicants' claimed notification method provides an efficient mechanism by which the relevant option-change handling code is identified. Applicants' claimed notification method, unlike *McLennan*, neither needs properly initialized options nor an explicit call to a method to ensure that options will not be missing whenever a mega-widget is created.

Attention is drawn to the foregoing distinctions, as articulated in amended claim 1 cited herein, which underscore Applicant's inventive elements of notifying objects within

a class hierarchy and computing mappings that are stored in a mapping data structure if no mapping was previously computed:

> 1. (Previously Presented) A method of processing data comprising:
>
> defining an object with an option data structure which supports references to option values without preallocation of memory space for the full option values, *wherein the object is instantiated from a class within a class hierarchy*; and
>
> notifying the object of a change in an option value of an option through change handlers identified by an option binding, the option binding being located by *first searching a mapping data structure for a previously computed mapping to the option binding and, if no mapping was previously computed, by then computing the mapping to the option binding and storing the mapping in the mapping data structure,*
>
> *wherein code for each of the change handlers for the option may be defined in different classes within the class inheritance hierarchy* (emphasis added).

Thus, Applicants submit that the data structure and notification mechanism supported by *McLennan* does not disclose the elements of the invention that are recited in the claims.

References to Option Values without Preallocation of Memory Space

In the *Final Office Action*, (p. 3), the Examiner further alleges that an alternative data structure, supported in the [incr Tk] language, allows values to be stored in strings or arrays as options associated with an instance object without preallocation of memory space for the full options values.

Applicants respectfully disagree. The [incr Tk] language does not disclose references to option values without preallocation of memory space. *McLennan*, based on the [incr Tk] languge, does not disclose references to option values without preallocation of memory space. Applicants respectfully point out that the Examiner's

basis for rejection was previously presented on page 4 of the *Final Office Action*, dated June 16, 2004. A *Response* and *Request for Continued Examination* was submitted on September 8, 2004, in which Applicants argued that *McLennan* does not disclose references to option values without preallocation of memory space on pages 2-5. In the subsequent *Office Action* on this matter, dated November 30, 2004, ***Applicants' argument was found persuasive and the rejection was withdrawn*** on page 2.

Additionally, to further clarify the state of the prior art, Applicants hereby amend the statement in the specification background to more clearly reflect the state of the prior art as exemplified by the [incr Tk] language. The amended statement now reads, "Such a data structure has, for example, been supported in the [incr Tk] language, which allows values to be stored in strings or arrays as options associated with an instance object." With this amendment, Applicants clarify that *the [incr Tk] language is an example of a language that does preallocate memory space*, as Applicants will show by the references cited herein.

Additional support for Applicants' position may also be found in the prosecution history for a related application (U.S.S.N. 09/760,031), which is a Continuation in Part of U.S.S.N. 09/672,562, filed on September 28, 2000, based on the same provisional application (U.S. Provisional Application No. 60/162,825) filed November 1, 1999. In the *Amendment and Response to Requirement for Information*, dated December 6, 2004 of the related application (U.S.S.N. 09/760,031), the same non-substantive amendment to the statement in the specification as cited above was made and the argument that *McLennan* does not disclose references to option values without preallocation of memory space was addressed on pages 21-26. Evidence that the

arguments therein were found to be persuasive may be found on page 15 of the subsequent *Final Office Action*, dated June 15, 2005, in which the same non-substantive amendment to the specification cited herein was accepted, and a previous objection to the specification was withdrawn.

To further clarify the assumption underlying the [incr TK] language regarding the preallocation of memory space, Applicants refer the Examiner to the following documents (both of which were previously disclosed in the *Applicants' Supplemental Information Disclosure* dated December 30, 2003) in support of Applicants' contention that the data structure supported by [incr Tk] does preallocate memory space for option values:

(1) Tcl Developer Xchange, *Tk Library Procedures - Tk_ConfigureWidget Manual Page*, at http://www.tcl.tk/man/tcl8.2.3/TkLib/ConfigWidg.htm (accessed October 5, 2006), 7 pages [hereinafter *Tk_ConfigureWidget manuall]*.

(2) sourceforge.net*, Archetype Base Class for [incr Tk]*, at http://incrtcl.sourceforge.net/itk/Archetype.html (accessed October 5, 2006), 4 pages [hereinafter *Archetype Base Class for [incr Tk]*].

Specifically, on page 6 of the printed document, *Tk_ConfigureWidget manual* describes the Tk_Offset macro:

> The Tk_Offset macro is provided as a safe way of generating the offset values for entries in Tk_ConfigSpec structures. It takes two arguments: the name of a type of record, and the name of a field in that record. It returns the byte offset of the named field in records of the given type.

Applicants submit that this statement suggests that every option value that can be set on a Tk widget has a unique, preallocated location in the widget's "record" where that option value will be stored. If space were allocated only for options that are actually set, it would not be possible to specify a fixed offset in the record for storing a value based only on knowledge of the type of record and the name of a field. It would also be

necessary to take into account the other options (if any) that had already been set on the specific record in question.

Further, on page 2 of the printed document, *Archetype Base Class for [incr Tk]* describes the "itk_component add" method:

> itk_component add name createCmds ?optionCmds?
>
> > Creates a component widget by executing the *createCmds* argument and registers the new component with the symbolic name *name.* The *createCmds* code can contain any number of commands, but it must return the window path name for the new component widget.
> >
> > The *optionCmds* script contains commands that describe how the configuration options for the new component should be integrated into the composite list for the mega-widget. It can contain any of the following commands:
>
> ***
>
> keep *option ?option option …?*
>
> > Integrates one or more configuration *options* into the composite list, keeping the name the same. Whenever the mega-widget option is configured, the new value is also applied to the current component. Options like "-background" and "-cursor" are commonly found on the keep list.
>
> ***
>
> If the *optionCmds* script is not specified, the usual option-handling commands associated with the class of the component widget are used by default.

Thus, the "itk_component add" command is used to add a component widget to a mega-widget. In calling this method, the "keep" command must be used to specify option names of the component widget that should be connected to option names of the mega-widget. Each option name to be "kept" is specified individually, and, subsequently, if one of these options is set on the mega-widget, the [incr Tk] system will set the same option value on the component widget.

Since each mega-widget instance will have its own collection of component widget instances, it follows that each mega-widget instance builds its own table of "kept" options, whose size is proportional to the number of options that could be set on the mega-widget, regardless of how many of those options are actually set on the mega-widget.

Further, on page 3 of the printed document, *Archetype Base Class for [incr Tk]* describes the "itk_option define" command:

> itk_option define *switchName resourceName resourceClass init ?config?*
>
> > This command is used at the level of the class definition to define a synthetic mega-widget option. Within the configure and cget methods, this option is referenced by *switchName*, which must start with a "-" sign. It can also be modified by setting values for *resourceName* and *resourceClass* in the X11 resource database. The *init* value string is used as a last resort to initialize the option if no other value can be used from an existing option, or queried from the X11 resource database. If any *config* code is specified, it is executed whenever the option is modified via the configure method. The *config* code can also be specified outside of the class definition via the configbody command.

Thus, the "itk_option define" command provides a mechanism for defining option names on mega-widgets that may not correspond to option names on any of the component widgets. Note that the value of the "init" argument may be "used as a last resort to initialize the option if no other value can be used from an existing option, or queried from the X11 resource database." That the option value can be "initialized" using the "init" argument discloses that a memory location to hold the option value is preallocated. Because this location exists, it must contain some value, so the init value may be used as a last resort for this purpose. If a preallocated location did not already exist, it could not be "initialized."

Evidence that ***the above argument was accepted*** may be found on page 15 of the *Final Office Action* dated June 15, 2005 of related case U.S.S.N. 09/760,031, wherein the objection to the non-substantive amendment to the specification was withdrawn.

Thus, Applicants submit that the data structure supported by the [incr Tk] does preallocate memory space for option values. Consequently, the skilled person would understand that *McLennan*, based on the [incr Tk] language, does not disclose without preallocation of memory space.

Prior Art Does Not Disclose At Least Three Element of the Claims in Issue

To properly establish that a prior art reference anticipates a claim under 35 U.S.C. § 102, each and every element of the claims in issue must be found, either expressly described or under principles of inherency, in the single prior art reference.

As shown above, independent claims 1, 10, 19, and 20, and depending claims therefrom, contain at least three elements that are patentably distinguishable from and not disclosed by *McLennan*:

- notifying the object, instantiated from a class <u>*within a class inheritance hierarchy*</u>, of a change in an option value by searching a mapping data structure,
- notifying the object through change handlers identified by an option binding located <u>*by first searching a mapping data structure and, if no mapping was previously computed, by then computing and storing the mapping*</u>; and

- *supporting references to option values without preallocation of memory space*.

For the reasons stated above, Applicants respectfully request the withdrawal of the rejections of independent claims 1, 10, 19, and 20, and depending claims therefrom, under on 35 U.S.C. § 102(b).

Rejections Under 35 U.S.C. § 103

In the *Final Office Action* mailed May 5, 2006, the Examiner rejected claims 2 and 11 as being unpatentable over *McLennan* in view of Li et al. (US Patent No. 5,943,496), hereinafter *Li*, and rejected claims 7-8 and 16-17 as being unpatentable over *McLennan* in view of Hostetter et al. ("Curl: A Gentle Slope Language for the Web," Spring, 1997), hereinafter *Hostetter*. Applicants respectfully traverse the rejection under 35 U.S.C. § 103(a) because the Examiner has not established a prima facie case of obviousness.

To establish a prima facie case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. See M.P.E.P. § 2143. Second, there must be a reasonable expectation of success. See M.P.E.P. § 2143.02. Finally, the prior art reference, or references when combined, must teach or suggest all the claim limitations. See M.P.E.P. § 2143.03.

Rejection of Claims 2 and 11

Applicants respectfully traverse the rejection of claims 2 and 11 because a prima facie case of obviousness has not been established based on *McLennan* and *Li*. The *Final Office Action* fails to establish a prima facie case of obviousness for at least the reason that the cited references, taken alone or combined, fail to teach each and every element of claims 2 and 11.

In particular, the combination of *McLennan* and *Li* fail to teach at least four elements of the invention recited in the claims:

- notifying the object instantiated from a class *within a class inheritance hierarchy* by searching a mapping data structure;
- notifying the object by *first searching a mapping data structure, and if no mapping was previously computed, by then computing and storing the mapping*;
- the *mapping data structure is a hash table*; and
- references to option values *without preallocation of memory space*.

The Mapping Data Structure

Applicants address the first three of the above elements in conjunction in response to the Examiner's contention (see *Final Office Action*, pp. 8-9) that the combination of *McLennan* and *Li* teaches "the mapping data structure is a hash table."

Applicants respectfully disagree. Applicants note that the first two elements, which pertain to the mechanism by which an object is notified of a change in an option value, as argued in the foregoing remarks, are absent in the teachings of *McLennan*. It is Applicants' position that *Li* also does not disclose the notification mechanism. Therefore, even assuming arguendo that the combination of *Li* and *McLennan* provide a mechanism whereby "the mapping data structure is a hash table," the combination would still not result in elements of the invention recited in the claims. Specifically, the combination of *McLennan* and *Li* fails to disclose at least *notifying the object instantiated from a class within a class inheritance hierarchy by searching a mapping data structure*, and *notifying the object by first searching a mapping data structure, and if no mapping was previously computed, by then computing and storing the mapping*.

Furthermore, it is Applicants' position that *Li does not teach the mapping data structure at all*, and therefore, certainly not that the mapping data structure is a hash table.

Cited prior art *Li* discusses identifying an object instance associated with a particular instance name. In *Li*, an object name table stores a name with a component object instance. *Li's* object name table is a hash table structure that merely associates a name with an object, which is consistent with traditional object-oriented techniques.

In contrast, Applicants' claimed mapping data structure, which in one particular embodiment is a hash table, stores mappings to option bindings.

Because *Li's* hash table does not store option binding mappings, *Li* does not teach or suggest Applicants' claimed mapping data structure. *Thus, the combination of McLennan and Li also does not teach or suggest at least that "the mapping data structure is a hash table."*

References to Option Values without Preallocation of Memory Space

As previously discussed, *McLennan* does not disclose references to option values without preallocation of memory space. It is Applicants' position that *Li* does not teach or suggest option bindings, and thus *Li* does not teach or suggest an object with an option data structure which supports references to option values without preallocation of memory space. The teachings in *Li* refer to Java, a traditional object-oriented programming language. It appears that *Li* is consistent with traditional object-oriented techniques. As a result, a preallocated field is provided for each option when an object is created, even if the option is never set on the object. Therefore, the

combination of *McLennan* and *Li* fails to teach at least the concept in Applicants' invention of *references to option values without preallocation of memory space*.

As a result of the foregoing arguments, the combination of *McLennan* and *Li* at least fails to teach four elements of Applicants' invention recited in the claims:

- notifying the object instantiated from a class *within a class inheritance hierarchy* by searching a mapping data structure;
- notifying the object by *first searching a mapping data structure, and if no mapping was previously computed, by then computing and storing the mapping*;
- the *mapping data structure is a hash table*; and
- references to option values *without preallocation of memory space*.

For the reasons set forth above, claims 2 and 11 are in condition for allowance. Applicants respectfully request the withdrawal of the rejection of claims 2 and 11 under 35 U.S.C. § 103(a).

Rejection of Claims 7-8 and 16-17

Applicants respectfully traverse the rejection of claims 7-8 and 16-17 because a prima facie case of obviousness has not been established based on *McLennan* and *Hostetter*. The *Final Office Action* fails to establish a prima facie case of obviousness for at least the reason that the cited references, taken alone or combined, fail to teach each and every element of claim 7-8 and 16-17. In particular, it is Applicants' position that the combination of *McLennan* and *Hostetter* fails to teach at least the concept of *notifying the object by first searching a mapping data structure and, if no mapping was*

previously computed, by then computing the mapping to the option binding and storing the mapping in the mapping data structure.

Cited prior art *Hostetter* discusses Curl, which is a language for creating web documents that can be used for text formatting, scripting simple interactions, and programming complex operations. In *Hostetter*, graphical objects can be built into hierarchies and an object can inherit attribute values ("properties") from its ancestors in a graphical hierarchy. Specifically, on page 8 of the printed document, *Hostetter* describes a property in a Graphic object:

> A property is a (name, value) binding and each Graphic object has an associated list of properties. *When the value of a property is required, an upward search of the template / instance tree is performed, starting with the current object*...
>
> *Notification of a change in a property's value is propagated through the tree which may, in turn, cause various reformatting operations to take place*. (emphasis added)

Applicants contend that *Hostetter's* notification method must determine the object's response to the notification by testing the name of the affected property against the names of all properties of interest to the object. Moreover, the type of an object is generally specified by means of a class hierarchy in which the object's class inherits from a succession of superclasses. Each of these superclasses can have an interest in one or more properties that affect the object, and therefore each superclass will override the notification method and further compare the name of the changed property against the names of the properties of interest to that superclass. Thus, *Hostetter's* notification processing and propagation of property changes becomes cumbersome and expensive if property changes are frequent.

In contrast, the present invention provides two data structures, an option data structure and a mapping data structure, that enable the relevant option-change handling code to be identified *efficiently* when an option-change notification occurs. The use of these data structures is an integral part of claim 1 in the present application and constitutes a useful and inventive advance over the techniques described in *Hostetter*.

Therefore, the combination of *McLennan* and *Hostetter* fails to teach at least *notifying objects by first searching a mapping data structure and, if no mapping was previously computed, by them computing the mapping to the option binding and storing the mapping in the mapping data structure* of Applicants' claim.

For the reasons set forth above, Applicants respectfully request the withdrawal of the rejection of claims 7-8 and 16-17 under 35 U.S.C. § 103(a).

Conclusion

In view of the foregoing amendments and remarks, Applicants respectfully request reconsideration of this application and the timely allowance of the pending claims.

Please grant any extensions of time required to enter this response and charge any additional required fees to our deposit account 06-0916.

Respectfully submitted,

FINNEGAN, HENDERSON, FARABOW, GARRETT & DUNNER, L.L.P.

Dated: November 3, 2006

By: Maura K. Moran
Maura Moran
Reg. No. 31,859